TRANSMITTAL OF APPEAL BRIEF (Large Entity)

Docket No.: T2147-906520

In Re Application Of: **Armand NACHEF**

Application No.	Filing Date	Examiner	Customer No.	Group Art Unit	Confirmation No.
09/582,755	11/3/00	Kuo Liang J. Tang	181	2191	2807

Invention: **METHOD FOR CONTROLLING A FUNCTION EXECUTABLE BY SPECIFIC COMMANDS TO DIFFERENT SOFTWARE TOOLS**

COMMISSIONER FOR PATENTS:

Transmitted herewith in triplicate is the Appeal Brief in this application, with respect to the Notice of Appeal filed on

The fee for filing this Appeal Brief is: **$500.00**

☒ A check in the amount of the fee is enclosed.

☐ The Director has already been authorized to charge fees in this application to a Deposit Account.

☒ The Director is hereby authorized to charge any fees which may be required, or credit any overpayment to Deposit Account No. **50-1165**

☐ Payment by credit card. Form PTO-2038 is attached.

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

Signature

Dated: **June 24, 2005**

Jason H. Vick, Reg. No. 45,285
Miles & Stockbridge
Customer No. 181

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to "Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450" [37 CFR 1.8(a)] on

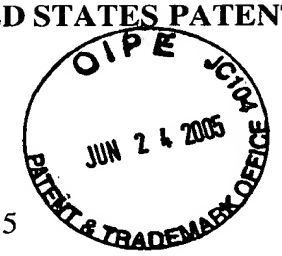
_____.
(Date)

Signature of Person Mailing Correspondence

Typed or Printed Name of Person Mailing Correspondence

cc: **jab**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE



In re application of :

Armand NACHEF et al. :

Application No.: 09/582,755 :

Filed: November 3, 2000 : Group Art Unit: 2122

For: METHOD FOR CONTROLLING A FUNCTION EXECUTABLE BY SPECIFIC COMMANDS TO DIFFERENT SOFTWARE TOOLS : Examiner: Kuo Liang J. TANG

* * *

APPEAL BRIEF

1. REAL PARTY IN INTEREST

The present application is assigned to Bull S.A. (France).

2. RELATED APPEALS AND INTERFERENCES

None known.

3. STATUS OF CLAIMS

Claims 7-27 are pending. Claims 7-27 stand rejected.

Specifically, claims 7-9, 11, 13, 15, 17, 19-20, 22-24 and 26-27 are rejected under 35 U.S.C. §103(a) as unpatentable over U.S. Patent No. 6,434,694 to Slaughter *et al.* (hereinafter "Slaughter") in view of U.S. Patent No. 5,634,016 to Steadham *et al.* (hereinafter "Steadham"). Furthermore, claims 8, 10, 12, 14, 16, 18, 21 and 25 are rejected under 35 U.S.C. §103(a) as unpatentable over Slaughter in view of Steadham and further in view of U.S. Patent No. 5,678,047 to Golshani *et al.* (hereinafter "Golshani").

4. STATUS OF AMENDMENTS

No amendments were filed after the Final Rejection. A Request for Reconsideration After Final was filed on March 23, 2005, however, the Advisory Action of April 22, 2005 indicated the Request for Reconsideration had been considered but did not please the application in condition for allowance.

5. SUMMARY OF THE INVENTION

The subject of the invention is a method for controlling a function executable by commands specific to different software products in a computer system. It is suitable for any system and is particularly adapted to a highly heterogeneous system. (Field of the Invention)

An object of the invention is to offer, in a computer system, a command common to all the commands representing the same function and specific to various software products, such as operating systems or applications. The subject of the invention is a method for controlling a function executable by various software products by means of commands specific to the respective software products and each capable of having at least one option, the software products being installed in at least one machine of a computer system, characterized in that it consists of defining in an abstract class an abstract method for the function, the abstract method including parameters corresponding to the union, in the logical sense, of all the options of the specific commands, of defining a common command that includes arbitrary symbols corresponding to the parameters of the abstract method, of creating at least one driver for implementing the abstract method in a machine, and of having the driver execute one of the specific commands with options equivalent to the options of the common command. The corollary subject of the invention is a computer system comprising at least one machine having various software products having in common at least one function executable by means of commands specific to the respective software products and

each capable of having at least one option, characterized in that it implements the method defined above. (Summary of the Invention)

The command interface 11 comprises a command module 12, a design module 13, an interface generator 14 and an interface base 15. The module 12 is connected by a two-way connection to the computer infrastructure 1. The interface base 15 is connected by a two-way connection to the modules 12 and 13 and to the generator 14. The generator 14 is also connected that it can be controlled by the module 13. The command module 12 is used by the user U to control and use the data of the system 10. The design module 13 is used by a designer C, who is another user of the interface 11 but who is a computer specialist. The two users U and C have respective consoles 16 and 17 attached to the respective modules 12 and 13. (Page 4)

The design module 13 also has a tool 50 for controlling similar functions in different operating systems and/or applications and a tool for automatically generating code 60 for implementing the interface tool 50. The computer system 10 represented in Fig. 1 will be considered to be a highly heterogeneous system, representing a case that is difficult to handle. In the example in question, the two operating systems 4a and 4b are two different versions of the UNIX operating system, known by the names Berkeley UNIX and AT&T UNIX. The two machines 2 are also assumed to include in the their respective memories 3 two printing software products that are different from one another 6a and 6b, such as the software products known by their registered trade names DPF, OpenSpool and XPRINT.

It is assumed that the two operating systems 4a, 4b and the two printing software products 6a, 6b have at least one common functionality F. However, since the software products 4a, 4b, 6a, 6b can be very different from one another, this functionality F can be executable by commands that are very different from one software product to another.

Several functionalities can be common to all or some of the software installed, such as the printing of documents and the saving of documents. The description will refer to the printing of documents as an example of the functionality F. This functionality can comprise several functions f, for example the functions: f1 "print"; f2 "list the current print jobs"; f3 "display the queues"; f4 purge the print jobs"; and f5 "change the priority of a print job in a queue." Hence, in this case, a functionality F is a family of functions f. The following example relates to the function f1 and will be enough to allow one skilled in the art to understand the invention overall.

It is assumed that the "print" function f1 in the four software products 4a, 4b, 6a and 6b in the computer system 10 of Fig. 1 is formed of four respective different specific commands: Pa and Pb for the respective operating systems 4a and 4b, and Pc and Pd for the printing software 6a and 6b. In the example of the software in question, the two different versions 4a and 4b of the UNIX operating system have respective print commands Pa = lpr" and Pb = lp.: However, it could also have print commands such as "enq" for the operating system known by the registered trade name AIX; "print" for the DOS operating system for personal computers; "mp" for the print software DPF OpenSpool; and "xpad" for the printing software known by the registered trade name XPRINT.

The command tool 50 is a software tool whose main object is to offer a user who is a computer expert, the designer C in Fig. 1, a common command P0 for the execution of each of the different print commands Pa-Pd in the example illustrated. As indicated in Fig. 1, the tool 50 illustrated comprises a block 51 for defining the common command P0, and two print drivers 52a, 52b for the two respective machines 2a and 2b, the drivers reacting to the common command P0 for generating the command Pa or Pc in the machine 2a and Pb or Pd in the machine 2b. In the example illustrated, the two drivers 52a and 52b are installed in the

two memories 3 of the two machines 2, while the block 51 is incorporated into the design module 13. The definition block 51 is connected to both drivers.

The command tool 50 implements the method for controlling a function f executable by means of different commands Pa-Pd in respective software products 4a, 4b, 6a, 6b, the commands each being capable of having at least one option. The method consists of defining in an abstract class an abstract method for the function f, the abstract method including parameters corresponding to the union, in the logical sense, of all the options (Table C) of the specific commands, of defining a common command (P0) that includes the arbitrary symbols corresponding to the parameters of the abstract method, of creating at least one driver (52) for implementing the abstract method in a machine, and of having the driver execute one of the specific commands with options equivalent to the options of the common command. (Pages 19-20)

Additional detailed description of exemplary embodiments of the invention can at least be found on pages 19-34 of the specification.

6. ISSUES

Whether the rejection of Claims 7, 9, 11, 13, 15, 17, 19-20, 22-24 and 26-27 under 35 U.S.C. §103(a) in view of Slaughter and Steadham should be reversed

Whether the rejection of Claims 8, 10, 12, 14, 16, 18, 21 and 25 under 35 U.S.C. §103(a) in view of Slaughter, Steadham and Golshani should be reversed.

7. GROUPING OF CLAIMS

The claims do not stand or fall together. The following groups of claims, and all of the other individual claims, are each separately patentable. The reasons for separate

patentability are set forth in the argument section *infra*. For clarity, the arguments made in support of the parent claims will not be repeated when arguing the merits of claims that depend therefrom, however are incorporated therein.

Group I	-	**Claims 7, 20 and 22**
Group II	-	**Claim 26**
Group III	-	**Claims 8 and 21**
Group IV	-	**Claims 9 and 10**
Group V	-	**Claims 11-14, 23 and 25**
Group VI	-	**Claims 15-19 and 24**
Group VII	-	**Claim 27**

8. ARGUMENT

8.1. The rejection of the claims in Group I in view of Slaughter and Steadham should be reversed.

Independent Claim 7 recites a method for controlling a function executable by various software products by means of commands specific to the respective software products and each command capable of having at least one option... including defining in an abstract class an abstract method for the function, the abstract method including parameters corresponding to a union, in the logical sense, of all options of a specific command, defining a common command that includes arbitrary symbols corresponding to parameters of the abstract method... creating at least one driver... and executing by the driver one of the specific commands....

Independent Claim 20 recites that each command is capable of having at least one option... and means for defining an abstract class and abstract method for the function,

the abstract method including parameters corresponding to a union, in the logical sense, of all the options of a specific command... means for creating at least one driver... and means for executing by the driver one of the specific commands....

One of the fundamental issues on Appeal is whether the cited references teach or suggest the claimed features. It has been the Office's position that this is the case while Appellants have argued that not only do the references fail to teach each and every claimed feature, but moreover that the Office is misinterpreting the teachings of the references and has failed to provide a legally supportable motivation supporting the combination of the references.

The Final Office Action states:

> As Per Claim 7, Slaughter disclosed:
> ***-defining in an abstract class an abstract method for the function, the abstract method including parameters corresponding to a specific command*** (see Column 6, Lines 27-37, "MainMemory 404 is an **abstract class** that includes with those attributes inherited from Memory 402 **abstract methods** for managing caching that are ultimately implemented in the instantiable classes PhysicalMemory 412, PortIOMemory 414, and VirtualMemory 416. The latter two classes inherit from MainMemory through the abstract class AccessibleMemory 410 that also inherits from MainMemory. Cache management methods are necessarily platform-specific; however, by using the abstract class MainMemory, those platform-specific memory management **functions** can be accessed in a platform independent manner.").
> ***-defining a common command that includes arbitrary symbols corresponding to parameters of the abstract method*** (see Column 6, Lines 27-37, "MainMemory 404 is an **abstract class** that includes with those attributes inherited from Memory 402 **abstract methods** for managing caching that are ultimately implemented in the instantiable classes PhysicalMemory 412, PortIOMemory 414, and VirtualMemory 416. The latter two classes inherit from MainMemory through the abstract class AccessibleMemory 410 that also inherits from MainMemory. Cache management methods are necessarily platform-specific; however, by using the abstract class MainMemory, those platform-specific memory management **functions** can be accessed in a platform independent manner.").
> ***-creating at least one driver for implementing the abstract method in a machine.*** (see Column 6, Lines 28-40, "In one embodiment, AccessibleMemory contains only platform-independent methods and is passed from bus managers to drivers."). *and*
> ***-executing by the driver one of the specific commands with options equivalent to the options of the common command*** (see Column 6, Lines 40-47, "Drivers also are configured to use only the platform-independent methods in

MainMemory and Memory. The platform-specific methods in PhysicalMemory, PortIOMemory, VirtualMemory, and DMAMemory are used by the bus manager, which has platform-specific information, to allow the driver to access memory in a platform-independent manner as described below.").

Slaughter discloses ***defining in an abstract class an abstract method for the function, the abstract method including parameters corresponding to a*** *specific* ***command*** (see Column 6, Lines 27-37, "MainMemory 404 is an abstract class that includes with those attributes inherited from Memory 402 abstract methods for managing caching that are ultimately implemented in the instantiable classes PhysicalMemory 412, PortIOMemory 414, and VirtualMemory 416. The latter two classes inherit from MainMemory through the abstract class AccessibleMemory 410 that also inherits from MainMemory. Cache management methods are necessarily platform-specific; however, by using the abstract class MainMemory, those platform-specific memory management functions can be accessed in a platform independent manner."). Slaughter does not explicitly disclose mapping the options of each specific command to the common command. However, Steadham teaches ***defining in an abstract class an abstract method for the function, the abstract method including parameters corresponding to a union, in the logical sense, of all options of a specific command (E.g.*** see FIG. 19 step 1902, 1904, 1906 and associated text)

Claim 7 recites that each command is capable of having at least one option and that the abstract method includes parameters corresponding to a union, in the logical sense, of *all* options of a specific command.

From a completely different technical field than the claimed invention, and directed toward solving an entirely different problem, Slaughter is directed toward a security system for a platform-independent device driver. While Slaughter discloses that "platform-specific memory management functions can be accessed in a platform independent manner," Slaughter is completely devoid of any teaching or suggestion that can be equated to the above feature.

Moreover, at no point does Slaughter teach or suggest that the abstract method can include parameters corresponding to a union of all the options of a specific command. While Slaughter discloses in various locations that the classes include objects, at no point does Slaughter teach or suggest the features of Claims 7 and 20.

Furthermore, as conceded by the Examiner, "Slaughter does not explicitly disclose mapping the options of each specific command to the common command." However, the Office Action pointed to Fig. 19 steps 1902, 1904 and 1906 of Steadham asserting that Steadham teaches "defining in an abstract class an abstract method for the function, the abstract method including parameters corresponding to the union and the logical since while the options of a specific command."

The Office maintained in the Final Office Action that "Steadham does teaches function SSINTER (E.G. Fig. 19, step 1906 and associated text) has all the options of function SSUNION (E.g. Fig 19, step 1902 an associated text) and function SSDIFF (E.g. Fig. 19, step 1904 and associated text). Therefore function SSINTER has all the union that contain all options of the selections sets (function SSUNION and SSDIF)." The April 22, 2005 Advisory Action then clarified that:

> As pointed out in the previous action dated 10/5/2004, the Examiner shows that it is Steadham who discloses the limitation, not by the Slaughter. (See page 5, lines 8-13).... As pointed out in the previous action dated 10/5/2004, the Eexaminer shows that Steadham does teach function SSINTER (e.g., Fig. 19, step 1906 and associated text) has all the options of function SSUNION (e.g., Fig. 19, step 1902 and associated text) and function SSDIFF (e.g., Fig. 19, step 1904 and associated text). Therefore, function SSINTER has all the union that contain all options of the selection sets (function SSUNION and SSDIFF). (See pages 2-3, Examiner's response).

For the Board's convenience, reproduced below is Fig. 19 of Steadham as well as the corresponding description found on column 33.

TABLE 4-continued

EVENT/CAD MENU STRUCTURE
[] Denotes program in [] to be called
() Description of simple AutoCAD command sequence

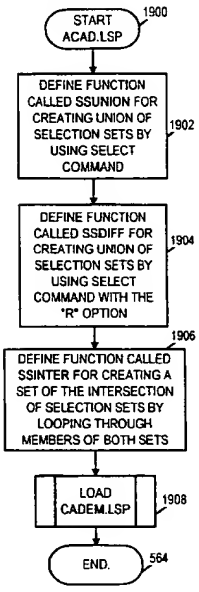
o Renumber Stage Modules	[dosort.lsp "b" nil nil 1 1 ll ur 1]
o UPDATE LEGEND	[legend.lsp]
o ASSIGN NAMES	(ddatte)
o MEASURE DISTANCE	[dist.lsp]
o CHAIR MAINTENANCE	[chairs.lsp]

When the Create or View/Exit options of the Drawing main menu selection are changed, the AutoCAD Portion of the EVENT/CAD Program module is called. That automatically executes the ACAD.LSP subroutine, the diagram of the flow chart which is shown in FIG. **19**. That subroutine sets the global variables in the EVENT/CAD module to their default values. It also sets the initial snaps and grid spacing. The subroutine then displays the main screen menu, creates a legend, and updates the status line area. The ACAD.LSP subroutine also defines several Lisp functions as well as determining whether or not the drawing is a FastAccess drawing.

When called, the ACAD.LSP subroutine starts at step **1900** and then defines a function called SSUNION for creating the union of selection sets by using the select command at step **1902**. It then defines a function called SSDIF at step **1904** for creating the union of selection sets by using the select command with the R option. The ACAD.LSP subroutine then defines a function called SSINTER at step **1906** for creating a set of the intersection of selection sets by looping through members of both sets. At step **1908**, the CADEM.LSP subroutine is loaded. The ACAD.LSP then ends at step **564**.

The CADEM.LSP subroutine is the second auto-executing AutoCAD function involved when the AutoCAD program is started. It defines Lisp functions to set global variables which store the plot-layout size (GETSIZE) and which reset the plot and title block size (SETSIZE). The CADEM.LSP subroutine also sets the global variables which are shown in Table 5.

FIG. 19



One of the key aspects argued during prosecution was whether the functions discussed on column 33 taught the claimed commands.

By way of background, Steadham is directed toward an event management system and more specifically to a computer integrated event management system designed for use by hotels and entertainment producers in hotels and other facilities in which banquettes, meetings, shows and other programs are held.

After a client has decided to book an event or meeting at a certain hotel or other facility, the final layout of each of the rooms which will be utilized for the client's event must be finalized. Based upon the layout of each of the rooms, which is, of course, dependent upon, among other things, the number of attendees, the type of meals, if any, to be served, the type and number of speakers or entertainment to be present at various times

during each of the meetings, etc., various inventory requirements must be met. Such inventory requirements include, among other items, the number and type of chairs, the number and type of tables, the size of any dance floor, the size of any stage, the number and types of podiums, stage modules, and followspot towers needed, as well as, for example, any special requirements, such as a movie screen or overhead projector. (Col. 1 of Steadham)

The present invention [Steadham] also allows the user to automatically generate and print room layouts in less time than that required to make a simple hand sketch. Each of the layouts is drawn to an exact scale so that there is no guesswork regarding how much space is left in the room or how many tables can be added. Since every element in the drawings can be controlled by the user, changes can be made to each event drawing, which are instantaneously reflected in the fully relational database. That serves to automatically update the information stored in the system relating to other functions, such as updating ECs, the available inventory and guest seating arrangements. (Col. 2, lines 20 *et seq.* of Steadham)

The relied upon portion of Steadham relates to the EVENT/CAD program module and how it works with three different command structures. In particular, the EVENT/CAD program module accomplishes the command structure by implementing many different programs written in the Autolisp® (Lisp) programming language and attaching them to a standard AutoCAD® structure.

As stated in Steadham, the EVENT/CAD program module, by providing unambiguous, menu-driven processing and a number of subroutines whose operation is transparent to the user, is user-friendly while both avoiding trivialization and retaining the immense power of a CAD-based designed systems.

Steadham discusses the ACAD.LISP subroutine in conjunction with steps 1900-1908 in Fig. 19, and defines functions called SSUNION, SSDIF and SSINTER. As discussed in greater detail below, there is simply no correlation between these functions, which are specific AutoCAD® routines and are used with inventory items, i.e., tables, staging, etc, (Steadham Col.30, lns. 10-48) and the features as set forth in the claims.

In particular, the "SSUNION" function, which is called by the ACAD.LSP subroutine, creates a "union of selection sets by using the select command at step 1902." The "selection sets" include inventory items such as tables, chairs, etc.

The SSDIF function creates a union of selection sets by using the select command with the R option and the SSINTER function creates a set of the intersection of selection sets. (See column 30-33 of Steadham).

While Steadham never actually provides an explicit definition of "selection sets," Appellants respectfully submit this a term of art in the computer assisted drafting environment. Specifically, AutoCAD® uses what is called a "selection set" to allow a user to group objects together, such as inventory items, within the computer assisted drafting environment and then to modify them. To support this assertion, Appellants have secured from the Autodesk® website (http://usa.autodesk.com), the makers of the Autocad® software, information (attached hereto) confirming our understanding that the definition of "selection sets" refers to selecting multiple objects in a computer aided design environment.

As is readily apparent, there is absolutely no correlation between the Autocad LISP subroutine relied upon by the Office and the features set forth in the claims.

It is well established law that three basic criteria must be met to establish a *prima facie* case of obviousness. All three of these criteria are lacking in the outstanding Office

Action. First, with respect to Claims 7 and 20, the references, taken either alone or in combination fail to teach each and every feature of the claims.

Secondly, since the references are from completely diverse technological fields, as evidenced by the drastically different types of problems being solved, there can be no expectation of success in their combination in that the asserted combination would alter the principle operation of each of the references.

Third, there is insufficient legally supportable motivation to combine the references. This is readily apparent in that the Office's relied upon motivation as stated on page 5 of the Final Office Action is entirely circular in that it alludes to modifying Steadham with the teachings of Steadham to "map the options of each specific command to the common command. The modification would have been obvious because one of ordinary skill in the art would have been motivated so that when the Create or View/Exit options of the Drawing mail menu selection are changed, the ACAD.LSP subroutine also defines several Lisp functions as well as determining whether or not the drawing is a FastAccess drawing..."

Based on the above deficiencies it is readily apparent that a *prima facie* case of obviousness has not been established. The rejection of the claims of Group I should thus be reversed.

8.2. The rejection of the claims in Group II in view of Slaughter and Steadham should be reversed.

Independent Claim 26 recites a method for controlling a function executable by various software products by means of commands specific to the respective software products and each command capable of having at least one option. The method comprises defining in an abstract class an abstract method for the function, the abstract method including parameters corresponding to all of the options of a specific command,

where the options are an argument that is capable of modifying the function of a specific command. The method further includes defining a common command that includes arbitrary symbols corresponding to parameters of the abstract method, creating at least one driver for implementing the abstract method in a machine and executing by the driver one of the specific commands with options equivalent to the options of the common command.

As is readily apparent from the above arguments made in relation to the claims of Group I, which are also relied up to traverse the rejection of Group II, the cited references simply fail to teach or suggest each and every feature of the claims. Specifically, the references, either alone or in combination at least fail to teach or suggest defining in an abstract class an abstract method for the function, the abstract method including parameters corresponding to all of the options of a specific command, the options are an argument that is capable of modifying the function of a specific command and creating at least one driver for implementing the abstract method in a machine and executing by the driver one of the specific commands with options equivalent to the options of the common command.

A *prima facie* case of obviousness has therefore not been established. The rejection of the claims of Group II should thus be reversed.

8.3. The rejection of the claims in Group III in view of Slaughter, Steadham and Golshani should be reversed.

The Final Office action concedes that "Slaughter and Steadham do not explicitly disclose creating a configuration file. However, Golshani teaches creating a configuration file...(see Column 2, Lines 30-34, "U2G also provides on-line help screens and explain pages and simulates a semi-UNIX-like environment by providing facilities for using shell variables and aliases. U2G supports I/O redirection and simple command

procedures, and simulates the piping of commands. A startup file, "u2gre," is first interpreted at the start of any session to set up the appropriate environment.")"

Appellants are entirely unclear as to how the passage relied upon by the Office has any bearing on the claimed features. Claim 8 recites creating a configuration file defining types and default values of the options of each specific command that can be executed by the driver, and determining parameters of one of said specific commands by consulting a configuration file by means of the common command.

Appellants have carefully reviewed Golshani and submit that the reference relates to a translator that can provide information about the translation and describes commands. In particular, upon selecting whether the UTG is to run in a "verbose" mode or a "terse" mode, the UTG is capable of providing information about the translation. However, being able to select an option for a translator and governing and its method of operation is not relevant to the claimed invention nor, is it combinable with the teachings of Slaughter or Steadham since each are from different fields of endeavor, are used to address different problems, would require a complete design of the Slaughter and Steadham inventions and the motivation to combine the teachings is lacking.

Based at least on these factors, and the deficiencies noted above in relation to the claims from which the subject claims depend, a *prima facie* case of obviousness has not been established.

In that comparable arguments can be made for Claim 21, the rejection of the claims in Group II is untenable and should be reversed.

8.4. The rejection of the claims in Group IV in view of Slaughter, Steadham and Golshani should be reversed.

In addition to the arguments made above in relation to the parent claims, Claims 9 and 10 recite that the "driver corresponds to a machine of the computer system." In that

Claims 9 and 10 depend, respectively, from Claims 7 and 8, the driver executes one of the specific commands with options equivalent to the options of the common command. The Office points to Slaughter asserting that the statement "(see Column 6, Lines 28-40, "In one embodiment, AccessibleMemory contains only platform-independent methods and is passed from the bus managers to drivers,")" renders obvious the claimed feature. After a careful review of the cited passage, Appellants can see no coloration between the claimed feature and the passage relied upon. A teaching or suggestion of the claimed feature is simply not there.

In that each and every feature is neither taught nor suggested by the cited references, the rejection is untenable and should be reversed.

8.5. The rejection of the claims in Group V in view of Slaughter, Steadham and Golshani should be reversed.

Claims 11-14 and 23 recite that the abstract class is the most abstract class that can be defined. Claim 25 recites that the abstract class is an interface in a programming language.

The Final Office Action does not include any specific indications of the passage(s) being relied upon for teaching or suggesting the above features. After a careful review of the cited references, it is respectfully asserted that the claimed features are entirely lacking therefrom. The rejection under 35 U.S.C §103 is thus untenable and should be reversed.

8.6. The rejection of the claims in Group VI in view of Slaughter, Steadham and Golshani should be reversed.

The claims in Group VI are generally directed toward specifying that that the abstract class contains at least some of the methods relating to functions of a functionality

common to the software products. The Office Action relies upon Col. 4, Lines 61-63 of Slaughter for this teaching.

Slaughter specifically states:

> Runtime system 208 includes, in one embodiment, a Java Virtual Machine ("JVM") 210 that receives instructions in the form of machine-independent bytecodes produced by the application running in applications layer 206 and interprets the instructions by converting and executing them.... Runtime system 208 further includes a set of additional functions 212 that support facilities such as I/O, network operations, graphics, printing, and the like. Also included with runtime system 208 is device interface 214 that supports the operation of buses 106 and 118, and devices 108, 110, and 112.

Appellants respectfully submit that neither this portion nor any other portion of the cited references teach or suggest the claimed feature. In contrast, there is simply no correlation between the claimed abstract class and the cited Java Virtual Machine and runtime system of Slaughter.

Since the cited references fail to teach or suggest each and every claimed feature, the rejection is untenable and should be reversed.

8.7. The rejection of the claims in Group VII in view of Slaughter and Steadham should be reversed.

Claim 27 recites that the options are an argument that is capable of modifying the function of the specific command.

The Final Office Action concedes that this is not disclosed by Slaughter but points to Steadham, and in particular:

> (e.g. FIG. 19 step 1904 and associated text, e.g. col. 33:26-31 which states "When Create or View/Exit options of the Drawing main menu selection are changed,... by using the select command with the R option.")" The Examiner's motivation for this teaching is that "it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of the Steadham into the system of Slaughter, so that *options are an argument that is capable of modifying the function of the specific command.* The modification would have been obvious because one of ordinary skill in the art would have been motivated so that the user can easily select different options to perform different actions (executed by the sceecific commands). (Emphasis Added)

Not only do the cited portions not teach or suggest the claimed feature, but the motivation relied upon to combine the teachings of Slaughter and Steadham is the exact feature being claimed.

In that the references, taken either alone or in combination, fail to teach or suggest every claimed feature, and the motivation supporting their combinability is fatally defective, the outstanding rejection is untenable and should be reversed.

9. CONCLUSION

Based on the foregoing, it is clear that a *prima facie* case of obviousness has not been established. It is respectfully requested the Final Rejection be reversed and the Application remanded to the Examiner for a prompt allowance.

The Commissioner is hereby authorized to charge to deposit account number 50-1165 for any fees not included herein that may be required by this paper and to credit any overpayment to the same Account. If any additional extension of time is required in connection with the filing of this paper and has not been separately requested, such extension is hereby petitioned.

Respectfully submitted,

Miles & Stockbridge P.C.

June 24, 2005
Date

Edward J. Kondracki
Reg. No. 20,604

Jason H. Vick
Reg. No. 45,285

Miles & Stockbridge P.C.
1751 Pinnacle Dr., Suite 500
McLean, VA 22102
Phone 703-903-9000
Fax 703-610-8686

APPENDIX

Listing of Claims:

1-6. (Cancelled)

7. A method for controlling a function executable by various software products by means of commands specific to the respective software products and each command capable of having at least one option, the software products being installed in at least one machine of a computer system, comprising defining in an abstract class an abstract method for the function, the abstract method including parameters corresponding to a union, in the logical sense, of all options of a specific command, defining a common command that includes arbitrary symbols corresponding to parameters of the abstract method, creating at least one driver for implementing the abstract method in a machine, and executing by the driver one of the specific commands with options equivalent to the options of the common command.

8. A method according to claim 7, wherein equivalence between options of the specific command and options of the common command comprises creating a configuration file defining types and default values of the options of each specific command that can be executed by the driver, and determining parameters of one of said specific commands by consulting a configuration file by means of the common command.

9. A method according to claim 7, wherein a driver corresponds to a machine of the computer system.

10. A method according to claim 8, wherein a driver corresponds to a machine of the computer system.

11. A method according to claim 7, wherein the abstract class is the most abstract class that can be defined.

12. A method according to claim 8, wherein the abstract class is the most abstract class that can be defined.

13. A method according to claim 9, wherein the abstract class is the most abstract class that can be defined.

14. A method according to claim 10, wherein the abstract class is the most abstract class that can be defined.

15. A method according to claim 7, wherein the abstract class contains at least some of the methods relating to functions of a functionality common to the software products.

16. A method according to claim 8, wherein the abstract class contains all or some of the methods relating to functions of a functionality common to the software products.

17. A method according to claim 9, wherein the abstract class contains all or some of the methods relating to functions of a functionality common to the software products.

18. A method according to claim 10, wherein the abstract class contains all or some of the methods relating to functions of a functionality common to the software products.

19. A method according to claim 11, wherein the abstract class contains all or some of the methods relating to functions of a functionality common to the software products.

20. A computer system comprising at least one machine having various software products having in common at least one function executable by means of commands specific to the respective software products and each command capable of having at least one option, and adapted to implement a method for controlling a function executable by various software products by means of commands specific to the respective software products and each command capable of having at least one option, the software products being installed in at least one machine of a computer system, means for defining in an abstract class an abstract method for the function, the abstract method including parameters corresponding to a union, in the logical sense, of all options of a specific command, means for defining a common command that includes arbitrary symbols corresponding to parameters of the abstract method, means for creating at least one driver for implementing the abstract method in a machine, and means for executing by the driver one of the specific commands with options equivalent to the options of the common command.

21. A computer system, according to claim 20, further comprising means for creating a configuration file defining the types and the default values of the options of each specific command that can be executed by the driver, and means determining the parameters

of one of said specific commands by consulting a configuration file by means of the common command so as to provide equivalence between the options of the specific command and the options of the common command.

22. A computer system according to claim 20 wherein a machine of the computer system includes a drive.

23. A computer system according to claim 20 wherein the abstract class is the most abstract class that can be defined.

24. A computer system according to claim 20 wherein the abstract class contains all or some of the methods relating to functions of a same functionality common to the software products.

25. A computer system as set forth in claim 23 wherein the abstract class is an interface in a programming language.

26. A method for controlling a function executable by various software products by means of commands specific to the respective software products and each command capable of having at least one option, comprising:

defining in an abstract class an abstract method for the function, the abstract method including parameters corresponding to all of the options of a specific command, where the options are an argument that is capable of modifying the function of a specific command;

defining a common command that includes arbitrary symbols corresponding to parameters of the abstract method;

creating at least one driver for implementing the abstract method in a machine; and

executing by the driver one of the specific commands with options equivalent to the options of the common command.

27. The method of claim 7, wherein the options are an argument that is capable of modifying the function of the specific command.

From http://usa.autodesk.com:

Autodesk

United States

Home | Products - Solutions - Subscription - Store - Support | A

Product Information
How to Buy
Autodesk Subscription
Consulting
Training
Custom Training
Authorized Training Centers
Courseware
How-to Articles
- Tutorials and How-to
- Expert Q&A
- CAD Management
- Design Collaboration & Internet
- Hardware & Systems
- Industry Trends
- Authors

Tips

Support
Data & Downloads

Autodesk VIZ

Autodesk VIZ: Saving Time Selecting Objects

By Nancy Fulton

Almost every editing operation you undertake in 3D Studio VIZ® software requires you to select objects. In this article we review the wide variety of tools 3D Studio VIZ provides for selecting objects, some techniques for creating named selection sets and groups, as well as tips for how to set up your designs so that you can later select objects more easily. By the time you complete this article you will have acquired skills that will make creating and maintaining your 3D Studio VIZ designs significantly easier.

Using a Mouse to Select Objects

You probably already know that you can select an object in 3D Studio VIZ just by clicking on it. But did you know that holding down the CTRL key lets you select multiple objects? If you select an object by accident, just hold down the ALT key and click it again to remove it from the set of selected objects.

> **Note:** When you select multiple objects in 3D Studio VIZ, you create a selection set. You can move, copy, scale, delete, and even apply modifiers to selection sets.

If you have to select a large number of objects, clicking them individually is time-consuming. The Selection/Xform toolbar contains tools that make it possible to select objects more efficiently. By default, this toolbar appears in the row of icons under the tab panel. If it's not visible on your screen, right-click on any toolbar and choose Selection/Xform. You can right-click on the toolbar to add it to the tab panel if desired. This makes it easy to find without cluttering up the interface.

If the Rectangular Selection Region icon on the Selection/Xform toolbar is enabled, you can select objects by creating a window around them (see Figure 1). If you select the Circular Selection Region icon, located under the Rectangular Selection Region icon on the flyout, selecting two points in a viewport creates a selection circle instead of a selection window. Selecting the Fence Selection Region icon lets you click a series of points that defines a selection boundary of any shape.

Using AutoLISP® to create selection sets

Published date: 2001-02-12
ID: TS25120

Applies to:
AutoCAD® 2002
AutoCAD® 2000i
AutoCAD® 2000

Issue

You want to create selection sets without using the GROUP command.

Solution

Instead of using the GROUP command to create a named selection set of objects, you can use AutoLISP® to assign a variable to a set of objects.

1. At the command prompt, type **(setq a (ssget))** and press ENTER (where 'a' is the AutoLISP variable).
2. At the Select Objects command prompt, select the objects to be assigned to the AutoLISP variable and press ENTER.

Whenever the Select Objects prompt is displayed, you can type **!a** and press ENTER to select the set you created.

You can create several sets by assigning a different variable name for each selection, which you can recall by typing **!<variable name>**.